UTM

UNIVERSITI TEKNOLOGI MALAYSIA

FACULTY OF COMPUTING
SEMESTER II, SESSION 2024/2025

BACHELOR OF COMPUTER SCIENCE (BIOINFORMATICS)

SECB3203 PROGRAMMING FOR BIOINFORMATICS - SECTION 01

PROJECT
ALZHEIMER'S DISEASE PREDICTION

GROUP MEMBERS	MATRIC NO
TAN ZHAO HONG	A23CS0188
CHIN PEI WEN	A23CS0065
KOO XUAN	A23CS0300

LECTURER'S NAME : DR. SEAH CHOON SEN

SUBMISSION DATE : 03 DECEMBER 2025

Table of Contents

3.0 Flowchart of the Proposed Approach

3.2 Model Development

For this project, classification was selected as the machine learning approach because the target variable, Alzheimer's disease diagnosis, is binary (0 = No Alzheimer's, 1 = Alzheimer's). Classification models are suitable for predicting discrete categories and are commonly used in medical diagnosis problems. Therefore, several classification algorithms were selected for model development and comparison.

3.2.1 Imported Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import (accuracy_score, precision_score, recall_score,
                             f1_score, confusion_matrix, classification_report,
                             roc_auc_score, roc_curve)
# from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
import warnings
warnings.filterwarnings('ignore')
```

Several Python libraries were imported to support data processing, model development, and visualization. Pandas and NumPy were used for data manipulation, Matplotlib and Seaborn for visualization, and Scikit-learn for model training, evaluation, and performance comparison.

3.2.2 Data Preparation for Model Development

```
    # SPLIT AND SCALE
    # Split the data
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=42, stratify=y
    )

    # Scale the features (Z-score normalization)
    print("\n--- Final Scaling (Z-Score Normalization) ---")
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    print(f"\nTraining set size: {X_train_scaled.shape}")
    print(f"Test set size: {X_test_scaled.shape}")

    print("\n" + "="*80)
    print("DATA PREPROCESSING COMPLETE")
    print("="*80)

    return X_train_scaled, X_test_scaled, y_train, y_test, X.columns
```

The dataset was divided into independent variables (features) and the dependent variable (diagnosis). The data was then split into training and testing sets using an 80:20 ratio. Feature scaling was applied using StandardScaler to normalize the data and improve model performance.

3.2.3 Classification models implemented for Alzheimer's disease prediction.

```
# ============================================================================
# 4. MODEL TRAINING AND EVALUATION
# ============================================================================

def train_and_evaluate_models(X_train, X_test, y_train, y_test):
    """Train multiple models and evaluate their performance"""

    # Define models
    models = {
        'Logistic Regression': LogisticRegression(random_state=42, max_iter=1000),
        'Decision Tree': DecisionTreeClassifier(random_state=42),
        'SVM': SVC(random_state=42, probability=True),
        'KNN': KNeighborsClassifier(n_neighbors=5)
    }

    results = {}
```

- **Logistic Regression**
  Logistic Regression was implemented as a baseline classification model. It estimates the probability of a patient having Alzheimer's disease based on the input features. This model is widely used in binary classification problems due to its simplicity and interpretability.

- **Decision Tree**
  Decision Tree was used to model decision rules in a tree-like structure. It is capable of capturing non-linear relationships and provides interpretability by visualizing decision paths.

- **Support Vector Machine (SVM)**
  Support Vector Machine (SVM) was implemented to find the optimal hyperplane that separates Alzheimer's and non-Alzheimer's cases. SVM is effective in high-dimensional spaces and performs well in medical classification tasks.

- **K-Nearest Neighbors (KNN)**
  K-Nearest Neighbors (KNN) classifies samples based on the majority class of the nearest neighbors. This model was included to compare instance-based learning with other classification techniques.

```
    print("\n" + "="*80)
    print("MODEL TRAINING AND EVALUATION")
    print("="*80)

    for name, model in models.items():
        print(f"\n{name}:")
        print("-" * 40)

        # Train the model
        model.fit(X_train, y_train)

        # Make predictions
        y_pred = model.predict(X_test)
        y_pred_proba = model.predict_proba(X_test)[:, 1] if hasattr(model, 'predict_proba') else None

        # Calculate metrics
        accuracy = accuracy_score(y_test, y_pred)
        precision = precision_score(y_test, y_pred, average='weighted')
        recall = recall_score(y_test, y_pred, average='weighted')
        f1 = f1_score(y_test, y_pred, average='weighted')
```

```
        # Store results
        results[name] = {
            'model': model,
            'accuracy': accuracy,
            'precision': precision,
            'recall': recall,
            'f1_score': f1,
            'y_pred': y_pred,
            'y_pred_proba': y_pred_proba
        }

        # Print results
        print(f"Accuracy:  {accuracy:.4f}")
        print(f"Precision: {precision:.4f}")
        print(f"Recall:    {recall:.4f}")
        print(f"F1-Score:  {f1:.4f}")

        # Cross-validation score
        cv_scores = cross_val_score(model, X_train, y_train, cv=5, scoring='accuracy')
        print(f"CV Accuracy: {cv_scores.mean():.4f} (+/- {cv_scores.std():.4f})")

    return results
```

Figure : Model Training Loop

Each model is trained using the fit() method on the training set, then predictions are made on the test set for evaluation.